EOF-Modeling-EOModeler

Table of Contents

Value type documentation

Yes it exists and it is here: WebObjects - WorkingWithAttributes

Class generation

This is one of the most vexed and difficult area. The main issue is the separation of generated code from your own business logic.

Here are some ways of dealing with this issue:

Apple suggested way

This is pretty dangerous as you will need to do a merge each time you do some modification to the model. I have never managed to make it work without a lot of sweat.

Inheritence

This is the method I use most of the time. For each real entity you create two entities in the model: one that will contain the business logic and one that will contain all the accessors. The accessor entity is an abstract entity an the pare n t of the real entity. The real entity is the one that you application will interact with and will never need to be regenerated. The abstract entity is used to carry all the accessor and you can regenerate it whenever it is needed as it does carry any of y our code.

All relationship must be established with the real entity or you will have problems

Just one more trick with this solution try to keep all the abstract entities together by prefixing them with a com mon pre fix. That way you can select all of them and regenerate them all with one command (nearly you still need to acknowledge the overwrite confirmation).

EOGenerator

This tool makes the inheritance trick transparent. For more information, read the EOGenerator section.

?

Although using EOGenerator or the other method above to model your EOs (using the GenerationGap design pattern) is helpful when working with EOModeler, it also means that you have double the number of EO classes to deal with, and more messaging up the inheritance hierarchy at runtime, and more files to check in/out of CVS, etc.

What would be nice would be if EOGenerator had the ability to 'collapse' the 2 related EO classes together once you decide that your model is relatively stable, and you won't be needing to make a lot of EOModel changes any more. Early on in the project lifecycle, when more changes are made, EOGenerator can be very useful.

Jesse Barnum

What about not generating class files from EOModeler at all (assuming you're subclassing EOGenericRecord), and leaving the EO classes empty except for custom methods, and accessors that you specifically manually add? This should work fine as long as you use KVC to get your object properties. Is there a downside to this that I'm not seeing?

By hand

Just add/remove/edit the attributes and relationships by hand.

Adding Referential Integrity Rules

You can use the Advanced Relationship Inspector to add referential integrity rules for a relationship.

To add referential integrity rules:

- Select the relationship for which you want to add rules.
- In the Relationship Inspector, click the Advanced Relationship Inspector icon.

You can use the fields in the Advanced Relationship Inspector to further specify a relationship. The options in this inspector are described in the following sections.

Batch Faulting

Normally when a fault is triggered, just that object (or array of objects for a to-many relationship) is fetched from the database. You can take advantage of this expensive round trip to the database by batching faults together. The value you type in the Batch Size field indicates the number of faults for the same relationship that should be triggered along with the first fault. For more discussion of batch faulting, see the class specification for EODatabaseContext in the Enterprise Objects Framework Reference.

Optionality

This field lets you specify whether a relationship is optional or mandatory. For example, you could require all departments to have a location (mandatory), but not require every employee to have a manager (optional).

Delete Rule

This field lets you specify the delete rules that should be applied to an entity that's involved in a relationship. For example, you could have a department with multiple employees. When a user tried to delete the department, you could:

- Delete the department and remove any back reference the employee has to the department (Nullify).
- Delete the department and all of the employees it contains (Cascade).
- Refuse the deletion if the department contains employees (Deny).
- Allow the deletion and do nothing to the destination objects (No Action).

The No Action rule is useful for tuning performance. However, you should use this delete rule with great caution since it can result in dangling references in your object graph. For more information, see the class specification for EOClassDescription in the Enterprise Objects Framework Reference.

Owns Destination

The Owns Destination checkbox lets you set a source object as owning its destination objects. When a source object owns its destination objects and you remove a destination object from the source object's relationship array, this also has the effect of deleting it from the database (alternatively, you can transfer it to a new owner). This is because ownership implies that the owned object can't exist without an owner-for example, line items can't exist outside of a purchase order.

Propagate Primary Key

The Propagate Primary Key checkbox lets you specify that the primary key of the source entity should be propagated to newly inserted objects in the destination of the relationship. This is typically used for an owning relationship, where the owned object has the same primary key as the source. For example, in the Movies database the TalentPhoto entity has the same primary key as the entity that owns it, Talent.

EOModeler Bugs

EOModeler is a very old application that has not gotten much attention from Apple since the NeXT purchase. There are quite a few odd behaviors, some of which are documented on the EOModeler Bugs page.

For EOModeler bugs related to Prototypes, see Prototype Gotchas.